AFRL-SR-AR-TR-10-0030

REPORT DOCUMENTATION PAGE

1. REPORT DATE *(DD-MM-YYYY)* 16-03-2010	2. REPORT TYPE Final Technical	3. DATES COVERED *(From - To)* May 2007 - November 2009

4. TITLE AND SUBTITLE
Scalable Simulations of Dynamics of Relationships

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-07-1-0437

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Gehrke, Johannes

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cornell University
373 Pine Tree Road
Ithaca, NY 14850

8. PERFORMING ORGANIZATION REPORT NUMBER
OSP 53525

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research RSL
875 N. Randolph Street, Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; distribution is Unlimited

20100511292

13. SUPPLEMENTARY NOTES

14. ABSTRACT
With funding from this grant, we developed a novel technique by which we can convert a simulation into a database query plan, and optimize its performance with database processing techniques. As part of this research we made the following five contributions: (1) We designed a novel design pattern for behavioral simulations; any simulation developed using this pattern can be converted into a database query plan. (2) We developed a novel language with formal semantics that supports this software design pattern. (3) We developed an optimizing compiler that automatically converts simulations written in our programming language into database query plans. (4) We developed novel algorithms for optimizing the query plans produced by our compiler. (5) We developed novel techniques for integrating our technology into a distributed simulation platform.

15. SUBJECT TERMS
Simulations, Programming Languages, Databases

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Johannes Gehrke
U	U	U	Unlimited		19b. TELEPHONE NUMBER *(Include area code)* 607-257-5438

Scalable Simulations of Dynamics of Relationships

Final Performance Report 2006-2009

PI: Johannes Gehrke
Award No: AFOSR FA9550-07-1-0437
Institution: Cornell University, Ithaca, NY 14853

Objectives

With funding from this project, we addressed the following research problems:

- **The application of database principles to large-scale behavioral simulations.** As the military expands its mission into new areas such as domestic disaster recovery, large-scale simulations of individual behavior are important tools for analyzing how best to address these situations. However, behavioral simulations are often complex, and do not scale enough to cover the population of a major urban area. How can we leverage the technology from massively scalable database systems to produce efficient large-scale behavioral simulations?
- **A language for expressing efficient large-scale behavioral simulations.** Defining a large-scale simulation in a database query language like SQL is prohibitively complex; there is no way to isolate or encapsulate individual behavior. What language features do we need to translate behavioral simulations of individuals into database query plans?
- **New algorithms for processing large-scale simulations.** The types of database query plans that correspond to large-scale simulations are likely very different from traditional database query plans. What new algorithms can we develop specifically to optimize these types of query plans?

Executive Summary

With funding from this grant, we were able to develop a novel technique by which simulations are converted into a database query plan, and optimize its performance with database processing techniques. As part of this research we made the following five contributions:

1. We designed a novel design pattern for behavioral simulations; any simulation developed using this pattern can be converted into a database query plan (either manually or automatically).
2. We developed a novel language with formal semantics that supports this software design pattern.
3. We developed an optimizing compiler that automatically converts simulations written in our programming language into database query plans.
4. We developed novel algorithms for optimizing the query plans produced by our compiler.

5. We developed novel techniques for integrating our technology into a distributed simulation platform.

Publications

- Walker White, Alan Demers, Christoph Koch, Johannes Gehrke, and Rajmohan Rajagopalan. Scaling Games to Epic Proportions. SIGMOD 2007: 31-42.
- Walker White, Benjamin Sowell, Johannes Gehrke, and Alan Demers. Declarative Processing for Computer Games. SIGGRAPH Sandbox 2008: 23-30.
- Walker White, Christoph Koch, Nitin Gupta, Johannes Gehrke, and Alan Demers. Database Research Opportunities in Computer Games. SIGMOD Record, September 2007, 7-13.
- Robert Albright, Alan Demers, Johannes Gehrke, Nitin Gupta, Hooyeon Lee, Rick Keilty, Gregory Sadowski, Ben Sowell, and Walker White. SGL: A Scalable Language for Data-Driven Games. SIGMOD 2008: 1217-1222.
- Nitin Gupta, Alan Demers, and Johannes Gehrke. SEMMO: A Scalable Engine for Massively Multiplayer Online Games. SIGMOD 2008: 1235-1238.
- Walker M. White, Christoph Koch, Johannes Gehrke, and Alan J. Demers. Better Scripts, Better Games. ACM Queue, Volume 6, Issue 7. November 2008.
- Walker M. White, Christoph Koch, Johannes Gehrke, Alan J. Demers: Better Scripts, Better Games. Communications of the ACM 52(3): 42-47 (2009)
- Nitin Gupta, Alan J. Demers, Johannes Gehrke, Philipp Unterbrunner, Walker M. White: Scalability for Virtual Worlds. ICDE 2009: 1311-1314
- Ben Sowell, Alan J. Demers, Johannes Gehrke, Nitin Gupta, Haoyuan Li, Walker M. White: From Declarative Languages to Declarative Processing in Computer Games. CIDR 2009
- Alan Demers, Johannes Gehrke, Christoph Koch, Benjamin Sowell, and Walker White. Database Research in Computer Games (Tutorial Session). SIGMOD 2009: 1011-1014.
- Marcos Vaz Salles, Tuan Cao, Benjamin Sowell, Alan Demers, Johannes Gehrke, Christoph Koch, and Walker White. An Evaluation of Checkpoint Recovery for Massively Multiplayer Online Games. VLDB 2009: 1258-1269.
- Scaling Simulations Through Declarative Processing. Microsoft eScience Workshop, Pittsburg, Pennsylvania, October 2009.

In addition, we have released a language specification document for the simulation language developed as part of this research. This document may be found online at

http://www.cs.cornell.edu/bigreddata/games/BRASILManual.pdf

Personnel

The following people worked on research associated with this award:

- Professor Johannes Gehrke was the PI of the project; he is an expert on the foundations of scalable systems. He worked on all aspects of the research.
- Dr. Walker White. White is a research associate; he is an expert on the theory of expressiveness and the design of languages. He worked on novel compiler optimizations and design patterns.
- Dr. Alan Demers is a principal scientist in Gehrke's group; he is an expert in the design and implementation of scalable systems. He worked on all aspects of the research.
- Professor Christoph Koch is on the faculty in the Department of Computer Science at Cornell University; he is an expert in declarative languages. He worked on the design of novel compilation techniques.
- Dr. Yanif Ahmad is a postdoc who worked on the implementation of scalable query processing algorithms.
- Dr. Marcos Vaz Salles is a postdoc who worked on techniques for scaling up simulations and on distributing simulation code.
- Iyer Parvati is a PhD student who worked on techniques for simulations on large graphs.
- Guozhang Wang is a PhD student who worked on techniques for scaling up simulations.

Interactions with and Transitions to Industry

In May 2008, Dr. Walker White made a visit to the ACES, the Flight Simulator product division at Microsoft, to present an overview of our technology. We also received additional funding from Microsoft for software development of a simulation engine. Walker White (co-PI) and Gehrke (PI) also gave various other talks at Microsoft about this work, and they are highly interested in the results.

Talks about Work Funded by This Award

- Scaling Games to Epic Proportions. Colloquium at the University of Texas at Austin. Austin, TX, March 2007.
- Scaling Games to Epic Proportions. 2007 RESCUE Distinguished Lecture Series, University of California Irvine, Donald Bren School of Information and Computer Science, March 2007.
- Scaling Games to Epic Proportions. Keynote at the Greater New York Area DB/IR Day Spring 2007. Hawthorne, NY, April 2007.
- Scaling Games to Epic Proportions. Colloquium at Microsoft Research. Seattle, WA, August 2007.
- Scaling Games to Epic Proportions. Colloquium at Yahoo! Research. Santa Clara, CA, August 2007.

- Scaling Games to Epic Proportions. Colloquium at the University of Massachusetts Amherst. Amherst, MA, April 2007.
- Scaling Computer Games to Epic Proportions. Undergraduate colloquium at the Dartmouth College. Hanover, New Hampshire, October 2007.
- Scaling Computer Games to Epic Proportions. Computer science colloquium at the Brown University. Providence, Rhode Island, November 2007.
- What's Next in Database Research? Colloquium at the University of Tromsø. Tromsø, Norway, December 2007.
- Scaling Games to Epic Proportions. Colloquium at the Department of Computer Science, ETH Zurich. Zurich, Switzerland, March 2008.
- Scaling Games to Epic Proportions. Distinguished Lecture at the Max Planck Institute for Software Systems. Saarbrucken, Germany, March 2008.
- Scaling Games to Epic Proportions. Colloquium at the Ecole Polytechnique Fédérale de Lausanne. Lausanne, Switzerland, March 2008.
- Applying Database Technology to Games and Simulations. Presentation to ACES, Microsoft Flight Simulator division. Redmond, Washington, May 2008.
- Scaling Games to Epic Proportions. Colloquium at the Humboldt-University at Berlin. Berlin, Germany, June 2008.
- Scaling Computer Games. Talk at Microsoft Research. Seattle, WA, July 2008.
- Scaling Games to Epic Proportions. Computer Science Distinguished Lecture Series, Department of Computer Science, University of Illinois at Urbana-Champaign. Urbana, IL, October 2008.
- Scaling Games to Epic Proportions. Colloquium at the University of Trondheim. Trondheim, Norway, December 2008.
- Declarative Processing for Computer Games. Distinguished Lecturer Series, Bren School of Information and Computer Sciences. Irvine, Ca, January 2009.
- Declarative Processing for Computer Games. Colloquium at the University of Washington, Department of Computer Science and Engineering. Seattle, WA, January 2009.
- Declarative Processing for Computer Games. Colloquium at Carnegie Mellon University, Department of Computer Science. Pittsburgh, PA, February 2009.
- Declarative Processing for Computer Games. Colloquium at RPI, Department of Computer Science. Troy, NY, February 2009.
- What Can Database Systems Do For Computer Games. Colloquium at Harvard University, Department of Computer Science. Boston, MA, April 2009.
- Bringing Database Research to Computer Games and Simulations. Keynote at the 35th International Conference on Very Large Data Bases. Lyon, France. August 2009.
- Scaling Simulations Through Declarative Processing. Microsoft eScience Workshop, Pittsburg, Pennsylvania, October 2009.
- A Database Approach to Scaling Games and Simulations. Colloquium at the Max Planck Institute for Software Systems. Saarbrucken, Germany, December 2009.
- A Database Approach to Scaling Games and Simulations. Colloquium at the Department of Computer Science, University of Heidelberg. Heidelberg, Germany, December 2009.

Major Discoveries

1. A novel programming pattern that allows behavioral simulations to be expressed as database query plans; from this pattern, we designed a novel programming language and an optimizing compiler.
2. Optimization algorithms for simulation query plans that provide asymptotically better performance over traditional techniques; this improvement is several orders of magnitude in many of our experiments.
3. Concurrency-control algorithms for processing distributed simulations.

Honors/Awards

Johannes Gehrke received a Faculty Development Award from the New York State Foundation for Science, Technology, and Innovation.

Major Technical Contributions

1. A Design Pattern for Efficient Simulated Behavior

A serious problem with large-scale behavioral simulations is their computational expense; as each individual may have to examine or interact with every other individual in order to determine its next action, the cost can grow quadratically with the number of individuals. However, much of this computation is redundant; individuals with similar properties see the world in the same way and come to the same conclusions.

This provides us with our motivation for transforming a simulation into a database query plan. Query plans process data *set-at-a-time*. That is, at every point of the computation, they read in all the data at once and perform related operations on it. This makes it easier to identify and eliminate redundant computation, and is a large part of what makes database management systems scalable.

However, database query languages are declarative languages. They do not have variables or assignments like we would find in an imperative language like C/C++ or Java. This is an issue for programming behavioral simulations. We would like to program each individual separately, and allow it to interact with others via the exchange of information. Exchanging information between programs is difficult without either variables or a message-passing framework, neither of which is present in declarative languages.

To address this problem, we have designed a new programming design pattern for behavioral simulations, which we call the *state-effect pattern.* This pattern allows individuals to exchange information via variable assignments; however, programs written in this pattern can always be rewritten to eliminate these variables. This makes it possible to rewrite the simulation in a database language like SQL

In the state-effect pattern, we assume that individual behaviors are processed together in discrete time steps. These time steps might correspond to animation frames or to some simulated unit of time (e.g. 1 second). Processing during each time step is separated into a *query phase* and an *update phase*. Intuitively, the query phase reads the state of an individual from a previous time step; it cannot make any changes to an individual. On the other hand, the update phase changes the state of an individual, but cannot access other individual while doing so. The advantage of these read-only and write-only phases is that they can be rewritten to use no variables at all; this makes the simulation more suitable for conversion into a database-like language.

To support these two phases, the attributes of each individual are classified as either *states* or *effects*. These classifications obey the following rules:

- State attributes are read-only in the query phase.
- Effect attributes are write-only in the query phase.
- Multiple writes to an effect attribute are *aggregated* via an aggregate function like sum or average.
- Effect attributes are read-only in the update phase.
- In the update phase, each individual updates its state attributes from its effects and old state values.

The key feature of this pattern is that effects are aggregated. An aggregation function takes in a set of data, and combines it into a single value; it does this independent of the order in which the data is received. This is an extremely powerful feature, as it allows us to dramatically rearrange the processing of individuals during the simulation. We can process individuals in any order, or even in parallel. We can split up an individual, processing it for a little while, switching to a new individual, and then returning to the original individual. All that matters is that all of the data for each effect attribute arrives before the end of the query phase.

To best apply this programming pattern, we need a custom programming language that enforces the pattern and transforms the program into a database query plan. However, we believe that the identification of this pattern is potentially useful for existing programming languages. While these languages may have features (such as variable assignments) that prevent their programs from being converted to a query plan, compiler extensions may be able to optimize the parts of their programs that conform to the state-effect pattern.

2. A Programming Language Supporting our Pattern for Efficient Simulated Behavior

In order to convert to the simulation to a database query plan, we need to ensure that it conforms to the state-effect pattern. For this reason, we have developed a programming language for behavioral simulations that makes the state effect pattern explicit. We call this language BRASIL (**B**ig **R**ed **A**gent **S**imulation **L**anguage). BRASIL is an object-oriented language that looks superficially like Java. Each behavior that we want to program is its own class; all individuals that implement this behavior are instances of this class.

Below is an example of a BRASIL script. This script represents a person performing a random walk through a crowd in two-dimensional space. The person uses an imaginary force to repel others nearby in order to push her way through the crowd.

```
class Person {
   // Location of person in 2D space
   public state float x : (x+vx);
   public state float y : (y+vy);

   // The latest velocity for the person
   public state float vx : vx + rand() + avoidx / count * vx;
   public state float vy : vy + rand() + avoidy / count * vy;

   // Used to update our velocity
   private effect float avoidx : sum;
   private effect float avoidy : sum;
   private effect int count : sum;

   /** The query-phase for this person. */
   public void run() {
      // Use "forces" to repel others that are too close
      foreach(Person p : Extent<Person>) {
         p.avoidx <- 1 / abs(x - p.x);
         p.avoidy <- 1 / abs(y - p.y);
         p.count  <- 1;
      }
   }}
```

There are a few important features to note about this script. First, all the fields are clearly denoted as either state or effect. Each state field has an update rule; this is an expression that takes effects and old state values and uses them to produce the new state value. In contrast, each effect field has an aggregation function associated with it. This function is used to combine all the values assigned to the same effect field. In this respect, effect fields are similar to aggregator variables in Google's Sawzall language; for this reason we use the Sawzall operator `<-` for assignments to effect fields.

Another thing to notice in this script is the `run()` method. This method defines the query phase for this individual. Intuitively, our simulation proceeds as follows:

- We process the run() method for each individual. This results in a collection of effect assignments.

- For each individual, we aggregate all of the effect assignments made to it in the query phase.
- For each state field of each individual, we apply the update rule to get the new value.

In order to convert our scripts into database query planes, we need place two further restrictions on our language. First of all, local values must also conform to the state-effect pattern, so that they can be eliminated as well. Local values are either constants, which cannot be changed after they are initialized, or effects, which have an associated aggregator. The following script is equivalent to our example above, but uses local values to for the main computation in the query phase:

```
class Person {
   // Location of person in 2D space
   public state float x : (x+vx);
   public state float y : (y+vy);

   // The latest velocity for the person
   public state float vx : nvx;
   public state float vy : nvy;

   // New velocity
   public effect float nvx : sum;
   public effect float nvy : sum;

   /** The query-phase for this person. */
   public void run() {
      // Local variables
      effect avoidx : sum;
      effect avoidy : sum;
      effect count  : sum;
      // Use "forces" to repel others that are too close
      foreach(Person p : Extent<Person>) {
         avoidx <- 1 / abs(x - p.x);
         avoidy <- 1 / abs(y - p.y);
         count  <- 1;
      }
      nvx <- vx + rand() + avoidx / count * vx;
      nvy <- vy + rand() + avoidy / count * vy;
}}
```

In addition to this restriction on local variables, we prohibit `for`-loops or `while`-loops. The only supported form of iteration is a `foreach`-loop, which iterates over the elements of a set; in the above example, this set is `Extent<Person>`, which is the set of all instances of the class `Person`. These loops function as a localized version of the state-effect pattern. Effect variables declared outside of a `foreach`-loop are write-only within the loop. However, effect variables can always be read outside of the `foreach`-loop, as that can be considered their update phase. In the above example, `avoidx` is write-only in the `foreach`-loop, but can be read for the computation of `nvx` outside of the loop.

While these limitations may seem restrictive at first, we have discovered that it is possible to design a large class of behavioral simulations within this language. Everything follows from the state-effect pattern; once that pattern is mastered, programming in BRASIL is relatively straightforward. When we combine this with the ability to compile BRASIL into database query plans, it becomes a very powerful framework for programming behavioral simulations.

Finally, we have integrated some features to support transactions for multiple effect fields. The state-effect pattern ensures that all individuals interact with one another simultaneously, and effects are aggregated together. When effects may be in conflict with one another, the aggregation method may simply pick a single effect and disregard the others. In essence, this is equivalent to aborting an assignment to an effect field. However, sometimes our simulation programs may want to couple two effects together. For example, in the first `Person` script above, we may want to assure that if an assignment to the `avoidx` effect field is aborted, then so is an assignment to the `avoidy` field. This coupling of assignments is very similar to that of transactions in databases, where an entire compound action is either committed or aborted.

3. An Optimizing Compiler That Converts BRASIL Programs into Database Query Plans

Because BRASIL strictly enforces the state-effect pattern, it is possible to compile any BRASIL script into a database query plan. This plan can be represented in either SQL or some other query plan representation format. In our work on BRASIL, we chose the relational algebra as our representation format, as it is common to all database query languages and is one of the cleanest representations.

Our compiler performs a source-to-source translation between BRASIL syntax and relational algebra expressions. The language features of BRASIL have a direct correspondence to database operations.
Conditionals correspond to selecting data from a database table, and `foreach`-loops are equivalent to joining two tables together. Assignments to an effect variable translate into aggregating data in a table.

One of the major advantages of converting BRASIL programs into database query plans is that we can use "rewrite rules" from the database literature to transform and reorder suboptimal code. For example, the program code

```
effect counts  : sum;
effect countd  : sum;
foreach(Person p : Extent<Person>) {
   foreach(Person q : Extent<Person>) {
      if (p == q) { counts <- 1; }
      else        { countd <- 1; }
   }
}
```

is equivalent to the program code

```
effect counts  : sum;
effect countd  : sum;
foreach(Person p : Extent<Person>) {
   counts <- 1;
}
```

```
    countd <- (counts-1)*(counts-1);
```

The first program has two nested loops and is much less efficient. When we convert these programs into the relational algebra, a database query processor will recognize that the joins in the first program are redundant and eliminate them. The resulting query plan after this join elimination will be the same as if we translated the second program directly.

There are many other rewrite rules that we can apply to the query plans generated by BRASIL scripts. Another example is using database duplicate elimination to remove redundant computation. In simple experiments where the individuals have to perform expensive scientific computation (but not much else), duplicate elimination has resulted in performance improvements of an order of magnitude.

4. Optimization Algorithms for Database Query Plans Representing Simulated Behavior

While rewrite rules allow us to perform some optimization on our behavioral simulations, they are primarily useful in removing unnecessary code. By themselves, they do not help with the quadratic behavior that results when each individual interacts with every other individual. To do that, we need to leverage other database techniques, such as automatic index generation.

By examining the query plans generated by the BRASIL compiler, we have seen that the vast majority of plans contain *spatial joins*, and that this is the source of the quadratic behavior. In a spatial join, each individual is paired with all of the other individuals nearby. There are many indices in the literature for improving database performance of spatial joins. However, these indices only improve performance if, for each individual, they can filter out a large number of individuals that do not interact with it. If the individuals in the simulation are close together, then none are filtered, and so there is no performance improvement.

However, we have discovered that we can improve performance even in the case where all individuals interact with one another, through the use of *aggregate indices*. Unlike in standard database query plans, the queries produced by BRASIL always follow a spatial join with an aggregate computation that collapses each set of pairings into a single value. Instead of filtering out extra individuals, these indices precompute the aggregate computation over sets of individuals so that we can share computation between multiple individuals.

Aggregate indexing is not new, but we can use it in novel ways by exploiting the guarantees provided by the state-effect pattern. In particular, the existence of separate query and update phases ensures there will be enough queries between index updates to amortize the cost of an asymptotically efficient bulk-build operation at every tick. Without this guarantee, we would need to use more conventional dynamic structures, supporting insertions and deletions, at a factor of log n penalty in asymptotic cost. In practice, this has resulted in an order of magnitude improvement on simulations with only 10000 individuals.

Other optimizations are enabled by the existence of query and update phases For example, we have been able introduce sweep-line algorithms from computational geometry into spatial indexing techniques to further improve the performance of our simulations.

5. Distribution of Database Query Plans Representing Simulated Behavior

Most of our work on this project has involved the use of database techniques to optimize behavioral simulations on a single machine. However, we have begun some initial investigations in using database techniques to distribute behavioral simulations across multiple machines. The primary issues are communication and synchronization: At the end of each time step, every individual has to receive the effect computations from all other individuals before proceeding. The necessary network communication can impede the performance of the simulation.

There are many techniques in the literature to reduce network overhead by limiting communication to individuals that are "visible" to one another. However, simple visibility is often not sufficient. For example, a simulation may have an "observer" object that gathers statistics from all other individuals in the database

Another issue with visibility-restricted algorithms is that they are unable to deal with the transitivity of actions; individuals can easily interact with one another even if they cannot see one another. Suppose we have three individuals *A*, *B*, and *C* where *A* and *B* are visible to each other, and *B* and *C* are visible to each other, but *A* and *C* cannot see one another. A visibility-restricted algorithm will not send any coordinated messages between *A* and *C*. As we have shown, this can lead to an inconsistent state in a distributed simulation.

Fortunately, when we translate simulations into database queries, they have a lot of semantic information that can be leveraged to limit communication. Simulations are essentially high-dimensional databases where the attributes can change only in predictable ways. By treating each time step as a database transaction, we were able to take ideas from distributed databases to limit communication overhead for a wide class of behavioral simulations. In particular, while were able to handle more complex behavioral interactions than the visibility-restricted algorithms could, our algorithms added only 1% overhead to the state-of-the-art visibility-restricted algorithms.

We also examined the issue of fault tolerance in distributed simulations. All of the individuals in a simulation are interdependent: If a machine in a distributed simulation goes down the simulation cannot proceed until it is restored. As part of this research we performed a thorough experimental evaluation of various checkpoint-recovery algorithms in the literature. Our analysis considered the overhead of supporting these algorithms and the expected recovery time. For our simulations, there was no clear winner. For simulations with very high update rates, a *naïve snapshot* scheme performs best, while for computationally intensive simulations with lower update rates the best option is a more sophisticated *copy-on-update* scheme.